Experiment No.8
Aim: To implement Restoring division algorithm using c-programming.
Name: AKSHAT AVINASH VYAS
Roll no: 61
Date of Performance:
Date of Submission:

Aim: To implement Restoring division algorithm using c-programming.

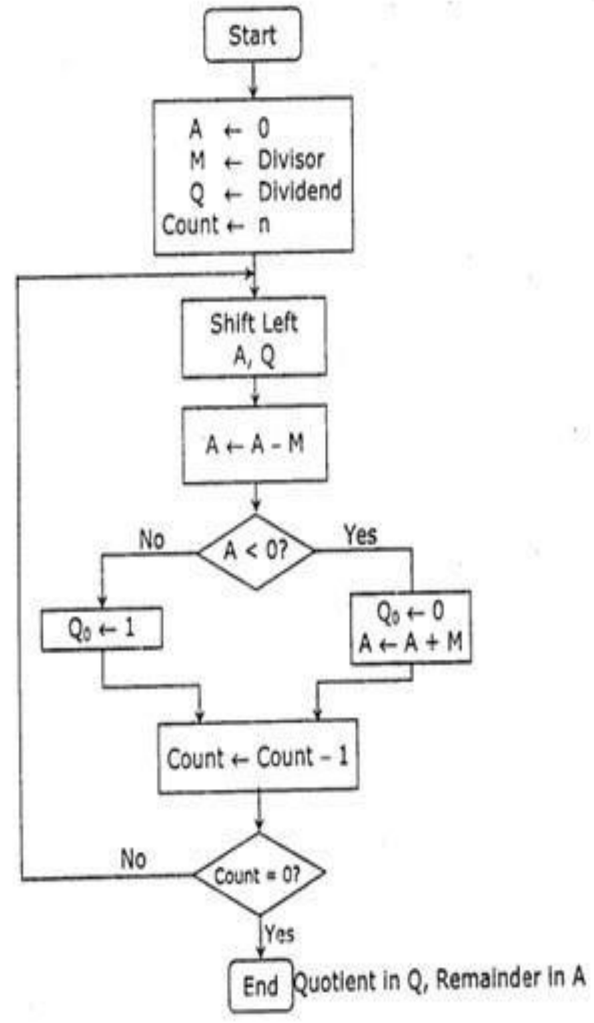
Objective -

1. To understand the working of Restoring division algorithm.
2. To understand how to implement Restoring division algorithm using c-programming.

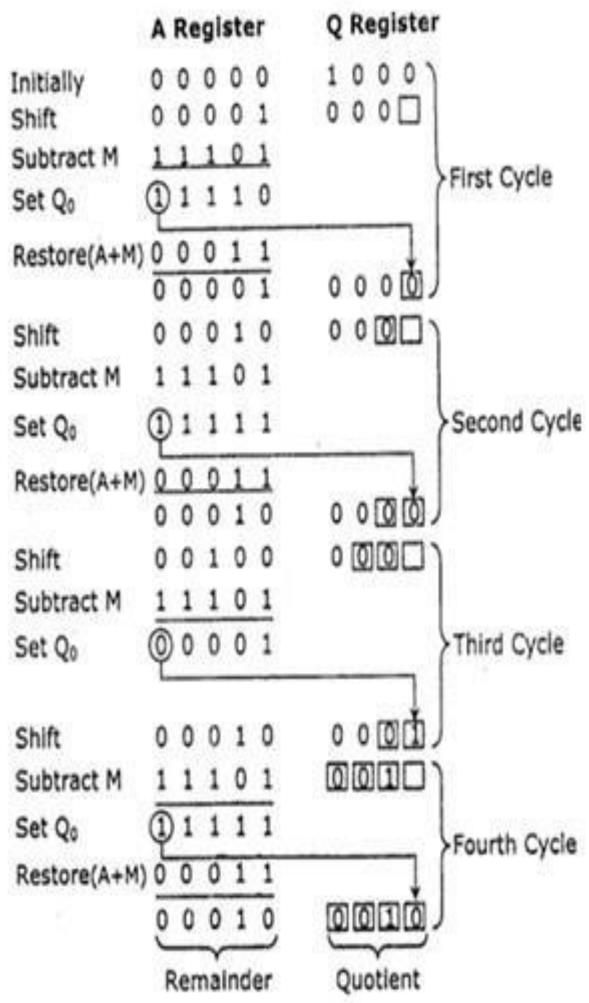
Theory:

1) The divisor is placed in M register, the dividend placed in Q register.
2) At every step, the A and Q registers together are shifted to the left by 1-bit
3) M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q0 set to 1-bit. Otherwise, Q0 gets a 0 bit and M must be added back to A to restore the previous value.
4) The count is then decremented and the process continues for n steps. At the end, the quotient is in the Q register and the remainder is in the A register.

Flowchart



Perform 8 ÷ 3 by restoring division technique.

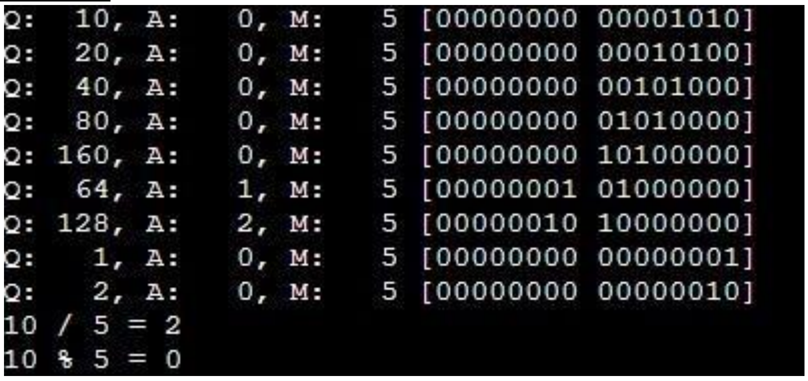


Program:

```
#include <stdio.h>

void binout_helper(int in, int left) {
 if (left == 0) return;
 binout_helper(in >> 1, left - 1);
  printf("%d",in & 1);
}

void binout(int in) {
 binout_helper(m,8);
}

void print_status(int Q,int A, int M) {
  printf("Q: %3d, A: %3d, M: %3d [",Q,A,M);  binout(A); printf(" ");
 binout(Q); printf("]\n");
}

int main() {
  unsigned char dividend = 10;
 unsigned char divisor = 5;


  unsigned char A = 0;
 unsigned char Q =
 dividend;  unsigned char M
 = divisor;
 print_status(Q,A,M);  for
 (int i = 0; i < 8; i++) {

   // Step 1: Shift A and Q
   left  int AQ = (A << 8) +
   Q;  AQ <<= 1;
   A = (AQ >> 8) & 0xFF;
   Q = AQ & 0xFF;

   // Step 2: Subtract M from A, place answer in A
   A -= M;

   if (A & (1 << 7)) {
     // If the sign bit of A is 1, set q_0 to 0
     Q &= ~1;
     // and add M back to A (restore A)
    A += M;
   } else {
     // otherwise, set q_0 to 1
    Q |= 1;
   }
```

```
    print_status(Q,A,M);
  }

  unsigned char quotient = Q;
 unsigned char remainder = A;
 printf("%d / %d = %d\n"
 ,dividend,divisor,quotient);
 printf("%d %% %d =
 %d\n",dividend,divisor,remainder);
 return 0;
}
```

Output:



```
Q:  10, A:   0, M:   5 [00000000 00001010]
Q:  20, A:   0, M:   5 [00000000 00010100]
Q:  40, A:   0, M:   5 [00000000 00101000]
Q:  80, A:   0, M:   5 [00000000 01010000]
Q: 160, A:   0, M:   5 [00000000 10100000]
Q:  64, A:   1, M:   5 [00000001 01000000]
Q: 128, A:   2, M:   5 [00000010 10000000]
Q:   1, A:   0, M:   5 [00000000 00000001]
Q:   2, A:   0, M:   5 [00000000 00000010]
10 / 5 = 2
10 % 5 = 0
```

Conclusion -

In conclusion, the binary restoring division using Booth's algorithm efficiently performs signed division in a binary system. By implementing this algorithm in C, we can effectively handle both positive and negative divisors and dividends, ensuring accurate results. The key steps include initializing registers, performing iterative shifts and additions based on the current quotient bit, and restoring the dividend as necessary. This method not only optimizes the division process but also enhances our understanding of binary arithmetic and algorithm design in programming. Overall, the C implementation of Booth's algorithm serves as a valuable educational tool for exploring advanced computational techniques.